EXPRESS MAIL NO. EV 161 879 152 US

Docket No. 99AB083-A

110003.96820

PATENT APPLICATION FOR

# DISTRIBUTED REAL-TIME OPERATING SYSTEM

by

Sivaram Balasubramanian

# DISTRIBUTED REAL-TIME OPERATING SYSTEM

## CROSS-REFERENCE TO RELATED APPLICATIONS

**[0001]** The present Application is a continuation-in-part of U.S. patent application No. 09/408,696 filed on September 30, 1999, and claims the benefit thereof.

## STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

--

## BACKGROUND OF THE INVENTION

**[0002]** The present invention relates to industrial controllers for controlling industrial processes and equipment and more generally to an operating system suitable for a distributed industrial control system having multiple processing nodes spatially separated about a factory or the like.

**[0003]** Industrial controllers are special purpose computers used for controlling industrial processes and manufacturing equipment. Under the direction of a stored control program the industrial controller examines a series of inputs reflecting the status of the controlled process and in response, adjusts a series of outputs controlling the industrial process. The inputs and outputs may be binary, that is on or off, or analog providing a value within a continuous range of values.

**[0004]** Centralized industrial controllers may receive electrical inputs from the controlled process through remote input/output (I/O) modules communicating with the industrial controller over a high-speed communication network. Outputs generated by the industrial controller are likewise transmitted over the network to the I/O circuits to be communicated to the controlled equipment. The network provides a simplified means of communicating signals over a factory environment without multiple wires and the attendant cost of installation.

**[0005]** Effective real-time control is provided by executing the control program repeatedly in high speed "scan" cycles. During each scan cycle each input is read and new outputs are computed. Together with the high-speed communications network, this

ensures the response of the control program to changes in the inputs and its generation of outputs will be rapid. All information is dealt with centrally by a well-characterized processor and communicated over a known communication network to yield predictable delay times critical to deterministic control.

**[0006]** The centralized industrial controller architecture, however, is not readily scalable, and with foreseeably large and complex control problems, unacceptable delays will result from the large amount of data that must be communicated to a central location and from the demands placed on the centralized processor. For this reason, it may be desirable to adopt a distributed control architecture in which multiple processors perform portions of the control program at spatially separate locations about the factory. By distributing the control, multiple processors may be brought to bear on the control problem reducing the burden on any individual processor and the amount of input and output data that must be transmitted.

**[0007]** Unfortunately, the distributed control model is not as well characterized as far as guaranteeing performance as is required for real-time control. Delay in the execution of a portion of the control program by one processor can be fatal to successful real-time execution of the control program, and because the demand for individual processor resources fluctuates, the potential for an unexpected overloading of a single processor is possible. This is particularly true when a number of different and independent application programs are executed on the distributed controller and where the application programs compete for the same set of physical hardware resources.

**[0008]** One weak point in the distributed control model is the introduction of communication delays in the execution of control tasks. These communication delays result from the need for different portions of the control program on different spatially separated hardware to communicate with each other. In a typical first-in/first-out (FIFO) communication system, where outbound messages are queued according to their time of arrival at the communication circuit, a message with a high priority, as may be necessary for the prompt completion of a control task, will always be transmitted later than an earlier arriving message of low priority. This can cause a form of unbounded priority inversion where low priority task block high priority tasks, and this may upset the timing requirements of the real-time control program.

**[0009]** A second problem with the distributed control model arises from operating distributed control devices in a multi-tasking mode to be shared among different program tasks. Such multi-tasking is necessary for efficient use of hardware resources. Present real-time multitasking operating systems allow the assignment of a priority to a given task. The user selects the necessary priority levels for each task to ensure that the timing constraints implicit in the real-time control process are realized.

**[0010]** One problem with this approach is first that it is necessarily conservative because the priorities must be set before the fact resulting in poor utilization of the scheduled resource. Further because the timing constraints are not explicit but only indirectly reflected in the priorities set by the user, the operating system is unable to detect a failure to meet the timing constraints during run time.

**[0011]** On the other hand, some dynamic scheduling systems (which adapt to the circumstances at run-time) exist but they don't accept user assigned priorities and thus provide no guarantee as to which tasks will fail under transient overload conditions. There are also scheduling systems for multi-tasking that allow for both setting of priorities and that have a dynamic component to allow for greater processor utilization, for example, those that use the Maximum Urgency First algorithm. See generally D. B. Stewart and P. K. Khosla, "Real Time Scheduling of Dynamically Reconfigurable Systems," Proceedings of the 1991 International Conference on Systems Engineering, Dayton August 1991 pp. 139-142.

**[0012]** Unfortunately, such algorithms require rescheduling of all tasks as a new task becomes ready for execution. This results in greater overhead and produces a potential for an unbounded number of context switches (in which the scheduled resource switches its task) which can be detrimental to guaranteeing a completion time for a particular task as required by real-time control. Further current scheduling systems do not provide any guarantee for execution time of the tasks and the potential allow low priority tasks to fail.

## SUMMARY OF THE INVENTION

**[0013]** In particular, the present invention relates to an interrupt manager for use in a distributed control system. The interrupt manager includes circuitry that (i) receives interrupt signals including a current interrupt, (ii) determines whether the current interrupt can be processed without delaying processing of a non-interrupt task beyond a

predetermined time, and (iii) inhibits, at least temporarily, processing of the current interrupt when it is determined that the processing of the current interrupt would delay processing of the non-interrupt task beyond the predetermined time.

**[0014]** The present invention additionally relates to a method of handling interrupts for use with a processor in a distributed control system. The method includes receiving a current interrupt signal, determining whether processing of the current interrupt signal would delay processing of a non-interrupt task beyond a predetermined time. The method further includes inhibiting, at least temporarily, the processing of the current interrupt signal when it is determined that the processing would delay the processing of the non-interrupt task beyond the predetermined time.

**[0015]** The present invention also relates to a method of scheduling messages being transmitted on a network among spatially-distributed control components of a distributed control system. The method includes receiving a message, receiving a relative timing constraint concerning the message, where the relative timing constraint is indicative of an amount of time, and inserting the message into a queue at a location that is a function of the relative timing constraint.

**[0016]** The present invention additionally relates to a method of coordinating a new control application program with other control application programs being performed on a distributed real-time operating system, where the distributed real-time operating system is for use with a control system having spatially separated control hardware resources. The method includes receiving the new control application program, and identifying control hardware resources from a resource list matching control hardware resources required by the new control application program. The method further includes allocating portions of a constraint associated with the new control application program to each identified control hardware resource, and determining whether the allocated portions of the constraint of the new control application program can be met while requirements of the other control application programs also are met.

**[0017]** The present invention further relates to a method of operating an application program on a distributed control system having a plurality of hardware resources. The method includes receiving high-level requirements concerning the application program, and determining low-level requirements based upon the high-level requirements. The method further includes allocating at least one of the high-level

requirements and the low-level requirements among at least some of the plurality of hardware resources, and operating the application program in accordance with the allocated requirements.

## BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

**[0018]** Fig. 1 is a simplified diagram of a distributed control system employing two end nodes and an intervening communication node and showing the processor, memory and communication resources for each node;

**[0019]** Fig. 2 is a block diagram showing the memory resources of each node of Fig. 1 as allocated to a distributed real-time operating system and different application programs;

**[0020]** Fig. 3 is an expanded block diagram of the distributed operating system of Fig. 2 such as includes an application list listing application programs to be executed by the distributed control system, a topology showing the topology of the connection of the hardware resources of the nodes of Fig. 1, a resource list detailing the allocation of the hardware resources to the application program and the statistics of their use by each of the application programs, and the executable distributed real-time operating system code;

**[0021]** Fig. 4 is a pictorial representation of a simplified application program attached to its high-level requirements;

**[0022]** Fig. 5 is a flow chart of the operation of the distributed real-time operating system code of Fig. 3 showing steps upon accepting a new application program to determine the low-level hardware resource requirements and to seek commitments from those hardware resources for the requirements of the new application program;

**[0023]** Fig. 6 is a detailed version of the flow chart of Fig. 5 showing the process of allocating low-level requirements to hardware resources;

**[0024]** Fig. 7 is a block diagram detailing the step of the flow chart of Fig. 5 of responding to requests for commitment of hardware resources;

**[0025]** Fig. 8a is a detailed view of the communication circuit of Fig. 1 showing a messaging queue together with a scheduler and a history table as may be implemented via an operating system and showing a message received by the communication circuit over the bus of Fig. 1;

**[0026]** Fig. 8b is a figure similar to that of Fig. 8a showing the scheduler of Fig. 8a as implemented for multi-tasking of the processors of Fig. 1;

**[0027]** Fig. 9 is a flow chart showing the steps of operation of enrolling the message of Fig. 8a or tasks of Fig. 8b into a queue;

**[0028]** Fig. 10 is a schematic representation of the interrupt handling system provided by the operating system and processor of Figs. 1 and 2; and

**[0029]** Fig. 11 is a flow chart showing the steps of operation of the interrupt handling system of Fig. 10.

## DETAILED DESCRIPTION OF THE INVENTION

### Distributed Control System

**[0030]** Referring now to Fig. 1, a distributed control system 10 includes multiple nodes 12a, 12b and 12c for executing a control program comprised of multiple applications. Control end nodes 12a and 12c include signal lines 14 communicating between the end nodes 12a and 12c and a portion of a controlled process 16a and 16b. Controlled process portions 16a and 16b may communicate by a physical process flow or other paths of communication indicated generally as dotted line 18.

**[0031]** In the present example, end node 12a may receive signals A and B from process 16a, and end node 12c may receive signal C from process 16b and provide as an output signal D to process 16b as part of a generalized control strategy.

**[0032]** End nodes 12a and 12c include interface circuitry 20a and 20c, respectively, communicating signals on signal lines 14 to internal buses 22a and 22c, respectively. The internal buses 22a and 22c may communicate with the hardware resources of memory 24a, processor 26a and communication card 28a (for end node 12a) and memory 24c, processor 26c, and network communication card 28c for end node 12c. Communication card 28a may communicate via network media 30a to a communication card 28b on node 12b which may communicate via internal bus 22b to memory 24b and processor 26b and to second network communication card 28b' connected to media 30b which in turn communicates with communication card 28c.

**[0033]** Generally during operation of distributed control system application programs are allocated between memories 24a, 24b and 24c to be executed on the respective nodes 12a, 12b and 12c with communications as necessary over links 30a and 30b. In an example control task, it may be desired to produce signal D upon the logical conjunction of signals A, B and C. In such a control task, a program in memory 24a would monitor signals A and B and send a message indicating both were true, or in this

example send a message indicating the state of signals A and B to node 12c via a path through communication cards 28a, 28b, 28b' and 28c.

**[0034]** A portion of the application program executed by processor 26c residing in memory 24c would detect the state of input C and compare it with the state of signals A and B in the received message to produce output signal D.

**[0035]** The proper execution of this simple distributed application program requires not only the allocation of the application program portions to the necessary nodes 12a, 12b and 12c, but prompt and reliable execution of those programs, the latter which requires the hardware resources of memory, processor, and communication networks 28a, 30a, 28b, 28b' 30b and 28c.

**[0036]** Referring now to Fig. 2 for this latter purpose, the distributed real-time operating system 32 of the present invention may be used such as may be centrally located in one node 12 or in keeping with the distributed nature of the control system distributed among the nodes 12a, 12b and 12c. In the latter case, the portions of the operating system 32 are stored in each of the memories 24a, 24b and 24c and intercommunicate to operate as a single system. In the preferred embodiment, a portion of the operating system 32 that provides a modeling of the hardware resources (as will be described) is located in the particular node 12a, 12b and 12c associated with those hardware resources. Thus, hardware resource of memory 24a in node 12a would be modeled by a portion of the operating system 32 held in memory 24a.

**[0037]** In addition to portions of the operating system 32, memory 24a, 24b and 24c include various application programs 34 or portions of those application programs 34 as may be allocated to their respective nodes.

Integrated Resource Management

**[0038]** Referring now to Fig. 3, the operating system 32 collectively provides a number of resources for ensuring proper operation of the distributed control system 10. First, an application list 36 lists the application programs 34 that have been accepted for execution by the distributed control system 10. Contained in the application list 36 are application identifiers 38 and high-level requirements 40 of the application programs as will be described below.

**[0039]** A hardware resource list 44 provides (as depicted in a first column) a comprehensive listing of each hardware resource of the distributed control system 10

indicating a quantitative measure of that resource. For example, for the principle hardware resources of processors 26, networks 31 and memories 24, quantitative measurements may be provided in terms of millions of instructions per second (MIPs) for processors 26, numbers of megabytes for memories 24 and megabaud bandwidth for networks. While these are the principal hardware resources and their measures, it will be understood that other hardware resources may also be enrolled in this first column and other units of measures may be used. Generally, the measures are of "bandwidth", a term encompassing both an indication of the amount of data and the frequency of occurrence of the data that must be processed.

**[0040]** A second column of the hardware resource list 44 provides an allocation of the quantitative measure of the resource of a particular row to one or more application programs from the application list 36 identified by an application name. The application name may match the application identifier 38 of the application list 36 and the indicated allocation quantitative measure will typically be a portion of the quantitative measure of the first column.

**[0041]** A third column of the hardware resource list 44 provides an actual usage of the hardware resource by the application program as may be obtained by collecting statistics during running of the application programs. This measure will be statistical in nature and may be given in the units of the quantitative measure for the hardware resource provided in the first column.

**[0042]** The operating system 32 also includes a topology map 42 indicating the connection of the nodes 12a, 12b and 12c through the network 31 and the location of the hardware resources of the hardware resource list 44 in that topology.

**[0043]** Finally, the operating system also includes an operating system code 48 such as may read the application list 36, the topology map 42, and the hardware resource list 44 to ensure proper operation of the distributed control system 10.

**[0044]** Referring now to Fig. 4, each application program enrolled in the application list 36 is associated with high-level requirements 40 which will be used by the operating system code 48. Generally, these high-level requirements 40 will be determined by the programmer based on the programmer's knowledge of the controlled process 16 and its requirements.

**[0045]** Thus, for the application described above with respect to Fig. 1, the application program 34 may include a single ladder rung 50 (shown in Fig. 4) providing for the logical ANDing of inputs A, B and C to produce an output D. The high-level requirements 40 would include hardware requirements for inputs and outputs A, B, C and D. The high-level requirements 40 may further include "completion-timing constraints" $t_1$ and indicating a constraint in execution time of the application program 34 needed for real-time control. Generally the completion-timing constraint is a maximum period of time that may elapse between occurrences of the last of inputs A, B and C to become logically true and the occurrence of the output signal D.

**[0046]** The high-level requirements 40 may also include a message size, in this case the size of a message AB which must be sent over the network 31, or this may be deduced automatically through use of the topology map 42 and an implicit allocation of the hardware.

**[0047]** Finally, the high-level requirements 40 include an "inter-arrival period" $t_2$ reflecting an assumption about the statistics of the controlled process 16a in demanding execution of the application program 34. As a practical matter the inter-arrival period $t_2$ need be no greater than the scanning period of the input circuitry 20a and 20c which may be less than the possible bandwidth of the signals A, B and C but which will provide acceptable real-time response.

**[0048]** Referring now to Fig. 5, the operating system code 48 ensures proper operation of the distributed control system 10 by checking that each new enrolled application program 34 will operate acceptably with the available hardware resources. Prior to any new application program 34 being added to the application list 36, the operating system code 48 intervenes so as to ensure the necessary hardware resources are available and to ensure that time guarantees may be provided for execution of the application program.

**[0049]** At process block 56, the operating system code 48 checks that the high-level requirements 40 have been identified for the application program. This identification may read a prepared file of the high-level requirements 40 or may solicit the programmer to input the necessary information about the high-level requirements 40 through a menu structure or the like, or may be semiautomatic involving a review of the application program 34 for its use of hardware resources and the like. As shown and

described above with respect to Fig. 4, principally four high-level requirements are anticipated that of hardware requirements, completion-timing constraints, message sizes, and the inter-arrival period. Other high-level requirements are possible including the need for remote system services, the type of priority of the application, etc.

**[0050]** Referring still to Fig. 5, as indicated by process block 58, the high-level requirements 40 are used to determine low-level requirements 60. These low-level requirements may be generally "bandwidths" of particular hardware components such as are listed in the first column of the hardware resource list 44. Generally, the low-level requirements will be a simple function of high-level requirements 40 and the objective characteristics of the application program 34, the function depending on a priori knowledge about the hardware resource. For example, the amount of memory will be a function of the application program size whereas, the network bandwidth will be a function of the message size and the inter-arrival period $t_2$, and the processor bandwidth will be a function of the application program size and the inter-arrival period $t_2$ as will be evident to those of ordinary skill in the art. As will be seen, it is not necessary that the computation of the low-level requirements 60 be precise so long as it is a conservative estimate of low-level resources required.

**[0051]** The distinction between high-level requirements 40 and low-level requirements 60 is not fixed and in fact some high-level requirements, for example message size, may in fact be treated as low-level requirements as deduced from the topology map 42 as has been described.

**[0052]** Once the low-level requirements 60 have been determined, at process block 62, they are allocated to particular hardware elements distributed in the control system 10. Referring also to Fig. 6, the process block 62 includes sub-process block 63 where the low-level requirements abstracted at process block 58 are received. At process block 66, end nodes 12a and 12c are identified based on their hardware links to inputs A, B and C and output D and a tentative allocation of the application program 34 to those nodes and an allocation of necessary processor bandwidth is made to these principal nodes 12a and 12c. Next at process block 68 with reference to the topology map 42, the intermediary node 12b is identified together with the necessary network 31 and an allocation is made of network space based on message size and the inter-arrival period.

**[0053]** The burden of storing and executing the application program is then divided at process block 70 allocating to each of memories 24a and 24c (and possibly 12b), a certain amount of space for the application program 34 and to processors 26a and 26c (and possibly 26b) a certain amount of their bandwidth for the execution of the portions of the application program 34 based on the size of the application program 34 and the inter-arrival period $t_2$. Network cards 28a, 28b', 28b and 28c also have allocations to them based on the message size and the inter-arrival period $t_2$. Thus, generally the allocation of the application program 34 can include intermediate nodes 12b serving as bridges and routers where no computation will take place. For this reason, instances or portions of the operating system code 48 will also be associated with each of these implicit hardware resources.

**[0054]** There are a large number of different allocative mechanisms, however, in the preferred embodiment the application program is divided according to the nodes associated with its inputs per United States Patent 5,896,289 to Struger issued Apr. 20, 1999 and entitled: "Output Weighted Partitioning Method for a Control Program in a Highly Distributed Control System" assigned to the same assignee as the present invention and hereby incorporated by reference.

**[0055]** During this allocation of the application program 34, the completion-timing constraint $t_1$ for the application program 34 is divided among the primary hardware to which the application program 34 is allocated and the implicit hardware used to provide for communication between the possibly separated portions of the application program 34. Thus, if the completion-timing constraint $t_1$ is nine milliseconds, a guaranty of time to produce an output after necessary input signals are received, then each node 12a-c will receive three microseconds of that allocation as a time obligation.

**[0056]** At process block 72, a request for a commitment based on this allocation including the allocated time obligations and other low-level requirements 60 is made to portions of the operating system code 48 associated with each hardware element.

**[0057]** At decision block 64, portions of the operating system code 48 associated with each node 12a-c and their hardware resources review the resources requested of them in processor, network, and memory bandwidth and the allocated time obligations and reports back as to whether those commitments may be made keeping within the allocated time obligation. If not, an error is reported at process block 66. Generally, it is

contemplated that code portions responsible for this determination will reside with the hardware resources which they allocate and thus may be provided with the necessary models of the hardware resources by the manufacturers.

**[0058]** This commitment process is generally represented by decision block 64 and is shown in more detail in Fig. 7 having a first process block 74 where a commitment request is received designating particular hardware resources and required bandwidths. At process block 76, the portion of the operating system code 48 associated with the hardware element allocates the necessary hardware portion from hardware resource list 44 possibly modeling it as shown in process block 78 with the other allocated resources of the resource list representing previously enrolled application programs 34 to see if the allocation can be made. In the case of the static resources such as memory, the allocation may simply be a checking of the hardware resource list 44 to see if sufficient memory is available. In dynamic resources such as the processors and the network, the modeling may determine whether scheduling may be performed such as will allow the necessary completion-timing constraints $t_1$ given the inter-arrival period $t_2$ of the particular application and other applications.

**[0059]** At the conclusion of the modeling and resource allocation including adjustments that may be necessary from the modeling at process block 80, a report is made back to the other components of the operating system code 48. If that report is that a commitment may be had for all hardware resources of the high-level requirements 40, then the program proceeds to process block 82 instead of process block 66 representing the error condition as has been described.

**[0060]** At process block 82, a master hardware resource list 44 is updated and the application program is enrolled in the application list 36 to run.

**[0061]** During execution of the application program 34 and as indicated by process block 84, statistics are collected on its actual bandwidth usage for the particular hardware resources to which it is assigned. These are stored in the third column of the hardware resource list 44 shown in Fig. 3 and is shown in the block 45 associated with Fig. 5 and may be used to change the amount of allocation to particular application programs 34, indicated by arrow 86, so as to improve hardware resource utilization.

Scheduled Communication Queuing

**[0062]** Referring now to Fig. 8a, the communication card 28 will typically include a message queue 90 into which messages 91 are placed prior to being transmitted via a receiver/transmitter 92 onto the network 31. A typical network queuing strategy of First-In-First-Out (FIFO) will introduce a variable delay in the transmission of messages caused by the amount of message traffic at any given time. Of particular importance, messages which require completion on a timely basis and which therefore have a high priority may nevertheless be queued behind lower level messages without time criticality. In such a queue 90, priority and time constraints are disregarded, therefore even if ample network bandwidth is available and suitable priority attached to messages 91 associated with control tasks, the completion timing constraints $t_1$ cannot be guaranteed.

**[0063]** To overcome this limitation, the communication card 28 of the present invention includes a queue-level scheduler 94 which may receive messages 91 and place them in the queue 90 in a desired order of execution that is independent of the arrival time of the message 91. The scheduler 94 receives the messages 91 and places them in the queue 90 and includes memory 98 holding a history of execution of messages identified to their tasks as will be described below. Generally the blocks of the queue 90, the scheduler 94 and the memory 98 are realized as a portion of the operating system 32, however, they may alternatively be realized as an application specific integrated circuit (ASIC) as will be understood in the art.

**[0064]** Each message 91 associated with an application program for which a time constraint exists (guaranteed tasks) to be transmitted by the communication card 28 will contain conventional message data 99 such as may include substantive data of the message and the routing information of the message necessary for transmission on the network 31. In addition, the message 91 will also include scheduling data 100 which may be physically attached to the message data 99 or associated with the message data 99 by the operating system 32.

**[0065]** The scheduling data 100 includes a user-assigned priority 96 generally indicating a high priority for messages associated with time critical tasks. The priority 96 is taken from the priority of the application program 34 of which the message 91 form a part and is determined prior to application program based on the importance of its control task as determined by the user.

**[0066]** The scheduling data 100 may also include an execution period (EP) indicating the length of time anticipated to be necessary to execute the message for transmission on the network 31 and a deadline period (DP) being in this case the portion of the completion timing constraint $t_1$ allocated to the particular communication card 28 for transmission of the message 91. The scheduling data 100 also includes a task identification (TID) identifying the particular message 91 to an application program 34 so that the high level requirements of the application program 34, imputed to the message 91 as will be described, may be determined from the application list 30 described above, and so that the resources and bandwidths allocated to the application program and its portion, held in resource list 44 can be accessed by the communication card 28 and the scheduler 94.

**[0067]** The scheduling data 100 may be attached by the operating system 32 and in the simplest case is derived from data entered by the control system programmer. The execution period after entry may be tracked by the operating system during run-time and modified based on that tracking to provide for accurate estimations of the execution period over time.

**[0068]** Upon arrival of a message at the communication card 28, the scheduling data 100 and the message data 99 are provided to the scheduler 94. The scheduler 94 notes the arrival time based on a system clock (not shown) and calculates a LATEST STARTING TIME for the message (LST) as equal to a deadline time minus the execution period. The deadline time is calculated as the message arrival time plus the deadline period provided in the message.

**[0069]** Referring now to Fig. 9, arrival of the message at the communication card 28 is indicated generally at process block 101 and is represented generally as a task, reflecting the fact that the same scheduling system may be used for other than messages as will be described below.

**[0070]** Following process block 101 is decision block 102 which determines whether the bandwidth limits for the task have been violated. The determination of bandwidth limits at block 102 considers, for example, the inter-arrival period $t_2$ for the messages 91. A message 91 will not be scheduled for transmission until the specified inter-arrival period $t_2$ expires for the previous transmission of the message 91. The expiration time of the inter-arrival period $t_2$ is stored in the history memory 98 identified

to the TID of the message. This ensures that all guarantees for message execution can be honored. More generally for a task other than a message, the bandwidth limits may include processor time or memory allocations.

**[0071]** If at process block 102, there is no remaining allocation of network bandwidth for the particular task and the task is guaranteed, it is not executed until the bandwidth again becomes available.

**[0072]** At succeeding block 104, if the bandwidth limits have not been violated, the message is placed in the queue 90 according to its user priority 96. Thus, high priority messages always precede low priority messages in the queue 90. The locking out of low priority messages is prevented by the fact that the high priority messages must have guaranteed bandwidths and a portion of the total bandwidth for each resource, the communication card 28, for example, is reserved for low priority tasks.

**[0073]** At decision block 106, it is determined whether there is a priority tie, meaning that there is another message 91 in the queue 90 with the same priority as the current message 91. If not, the current message 91 is enrolled in the queue 90 and its position need not be recalculated although its relative location in the queue 90 may change as additional messages are enrolled.

**[0074]** If at decision block 106 there is a priority tie, the scheduler 94 proceeds to process block 108 and the messages with identical priorities are examined to determine which has the earliest LATEST STARTING TIME. The LATEST STARTING TIME as described above is an absolute time value indicating when the task must be started. As described above the LATEST STARTING TIME need only be computed once and therefore doesn't cause unbounded numbers of context switches. The current message is placed in order among the message of a similar priority according to the LATEST STARTING TIME with earliest LATEST STARTING TIME first.

**[0075]** If at succeeding process block 110, there is no tie between the LATEST STARTING TIMES, then the enrollment process is complete. Otherwise, the scheduler 94 proceeds to process block 112 and the messages are examined to determine their deadline periods DP as contained in the scheduling data 100. A task with a shorter deadline period is accorded the higher priority in the queue 90 on the rationale that shorter deadline periods indicate relative urgency.

**[0076]** At succeeding process block 114 if there remains a tie according to the above criteria between messages 91 then at process block 116, the tie is broken according to the execution period, EP, of the messages 91. Here the rationale is that in the case of transient overload, executing the task with the shortest execution period will ensure execution of the greatest number of tasks.

**[0077]** A system clock with sufficient resolution will prevent a tie beyond this point by ensuring that the LATEST STARTING TIMES are highly distinct.

**[0078]** These steps of determining priority may be simplified by concatenating the relevant scheduling data 100 into a single binary value of sufficient length. The user priority forms the most significant bits of this value and the execution period the least significant bits. This binary value may then be examined to place the messages (or tasks) in the queue 90.

**[0079]** As each message 91 rises to the top of the queue 90 for transmission, its LATEST STARTING TIME is examined to see if it has been satisfied. Failure of the task to execute in a timely fashion may be readily determined and reported.

Mixed Priority Multi-Tasking

**[0080]** As mentioned, the scheduling system used for the communication card 28 described above is equally applicable to scheduling other resources within the distributed operating system, for example, the processors 26. Referring to Fig. 8b, each processor 26 may be associated with a task queue 119 being substantially identical to the message queue 90 except that each slot in the task queue 119 may represent a particular bandwidth or time slice of processor usage. In this way, enrolling a task in the task list not only determines the order of execution but allocates a particular amount of processor resources to that task. New tasks are received again by a scheduler 94 retaining a history of the execution of the task according to task identification (TID) in memory 98 and enrolling the tasks in one of the time slots of the task queue 119 to be forwarded to the processor 26 at the appropriate moment. The tasks include similar tasks scheduling data as shown in Fig. 8a but need not include a message data 99 and may rely on the TID to identify the task implicitly without the need for copying the task into a message for actual transmission.

**[0081]** Referring to Fig. 9, the operation of the scheduler 94 as with the case of messages above only allocates to the task the number of time slots in the queue 90 as was

reserved in its bandwidth allocation in the resource list 44. In this way, it can be assured that time guarantees may be enforced by the operating system.

Interrupt Management

**[0082]** As is understood in the art, interrupts normally act directly on the processor 26 to cause the processor 26 to interrupt execution of a current task and to jump to an interrupt subroutine and execute that subroutine to completion before returning to the task that was interrupted. The interrupt process involves changing the value of the program counter to the interrupt vector and saving the necessary stack and registers to allow resumption of the interrupt routine upon completion. Typically interrupt signals may be masked by software instructions such as may be utilized by the operating system in realizing the mechanism to be described now.

**[0083]** Referring now to Figs. 8a and 8b, a similar problem to that described above, of lower priority messages blocking the execution of higher priority messages in the message queue 90, may occur with interrupts. For example, a system may be executing a time critical user task when a low priority interrupt, such as that which may occur upon receipt of low priority messages, may occur. Since interrupts are serviced implicitly at a high priority level, the interrupt effects a priority inversion with the high priority task waiting for the low priority task. If many interrupts occur, the high priority tasks may miss its time guarantee.

**[0084]** This priority-inversion problem can be solved in a number of ways. Generally speaking, circuitry can be employed that receives interrupts and, upon receiving an interrupt, determines whether responding to the current interrupt would delay the execution of other tasks, particularly non-interrupt tasks, in a manner that would be excessive in terms of delaying the execution of the other tasks beyond a predetermined time. Various measures and techniques can be utilized to determine whether responding to the current interrupt would excessively delay the execution of other tasks. For example, the circuitry can determine whether the number of interrupts that have been processed recently, or are in queue to be processed (e.g., an interrupt that was just received, interrupts that have been received since a particular time, or interrupts that have been received but have not yet been processed), exceeds a certain maximum number. That maximum number can be, but need not be, associated with a particular period of

time. For example, the maximum number can represent a maximum number of interrupts that can be performed within a given amount of time.

**[0085]** Alternatively, a determination can be made whether the current interrupt satisfies a particular characteristic, such as a priority characteristic. For example, referring to Fig. 8a, upon a receipt of a message from network 31, an interrupt 118 may be generated and passed to a task generator 120 shown in Fig. 8b. The task generator 120 which receives the interrupt generates a proxy task forwarded to the scheduler 94. The proxy task assumes the scheduling data 100 of the message causing the interrupt and is subject to the same mixed processing as the tasks described above via the scheduler 94. Depending on its priority and other scheduling data 100, the proxy task may preempt the current task or might wait its turn. This procedure guarantees deterministic packet reception without affecting tasks on the receiving node adversely.

**[0086]** Alternatively, a determination can be made whether processing of the current interrupt will be accomplished in a manner satisfying a particular time constraint. For example, referring now to Fig. 10 in an alternate form of interrupt management, interrupts 118 from general sources such as communication ports and other external devices are received by an interrupt manager 122 prior to invoking the interrupt hardware on the processor 26. One exception to this is the timer interrupt 118' which provides a regular timer "click" for the system clock which, as described above, is used by the scheduler 94. The interrupt manager 122 provides a masking line 124 to a interrupt storage register 123, the masking line allowing the interrupt manager 122 to mask or block other interrupts (while storing them for later acceptance) and communicates with an interrupt window timer 126 which is periodically reset by a clock 127. Generally, the interrupt manager 122, its masking line 124, the interrupt storage register 123, the interrupt window timer 126 and the window timer are realized by the operating system 32 but as will be understood in the art may also be implemented by discrete circuitry such as an application specific integrated circuit (ASIC).

**[0087]** Referring to Fig. 11, the interrupt manager 122 operates so that upon the occurrence of an interrupt as indicated by process block 129, all further interrupts are masked as indicated by process block 128. The interrupt window timer 126 is then checked to see if a pre-allocated window of time for processing interrupts (the interrupt window) has been exhausted. The interrupt window is a percentage of processing time or

bandwidth of processor 26 reserved for interrupts and its exact value will depend on a number of variables such as processor speed, the number of external interrupts expected and how long interrupts take to be serviced and is selected by the control system programmer. In the allocation of processor resources described above, the interrupt period is subtracted out prior to allocation to the various application programs. The interrupt window timer 126 is reset to its full value on a periodic basis by the clock 127 so as to implement the appropriate percentage of processing time.

**[0088]** At process block 130, after the masking of the interrupts at process block 128, the interrupt window timer 126 is checked to see if the amount of remaining interrupt window is sufficient to allow processing of the current interrupt based on its expected execution period. The execution periods may be entered by the control system programmer and keyed to the interrupt type and number. If sufficient time remains in the interrupt window, the execution period is subtracted from the interrupt window and, as determined by decision block 132, then the interrupt manager 122 proceeds to process block 134. At process block 134, the interrupts 118 are re-enabled via masking line 124 and at process block 136, the current interrupt is processed. By re-enabling the interrupts at process block 134, nested interrupts may occur which may also be subject to the processing described with respect to process block 129. If at decision block 132, there is inadequate time left in the interrupt window, then the interrupt manager 122 proceeds to decision block 138 where it remains until the interrupt window is reset by the clock 127. At that time, process blocks 134 and 136 may be executed. As mentioned, the interrupt window is subtracted from the bandwidth of the processor 26 that may be allocated to user tasks and therefore the allocation of bandwidth for guaranteeing the execution of user tasks is done under the assumption that the full interrupt window will be used by interrupts taking the highest priority. In this way, interrupts may be executed within the interrupt window without affecting guarantees for task execution.

**[0089]** Although, in the above-described embodiment, a determination is made whether processing of the current interrupt can be completed within a time window, in other embodiments a decision as to whether to process the current interrupt can be based upon whether processing of the current interrupt will satisfy other time constraints. For example, in one embodiment, a current interrupt would be processed so long as processing could begin within a set time window. In another embodiment, a current

interrupt would be processed so long as processing of the current interrupt did not result in the violation of one or more completion timing constraints or other high-level or low-level requirements.

**[0090]** The above description has been that of a preferred embodiment of the present invention. It will occur to those that practice the art that many modifications may be made without departing from the spirit and scope of the invention. In order to apprise the public of the various embodiments that may fall within the scope of the invention, the following claims are made.